ticket is created with fewer uses than the first ticket, and the number of uses of the derived ticket is added to the use count of the first ticket, the owners of both tickets then know exactly how many uses their tickets are good for. We call tickets with this kind of individual number of uses *unshared tickets*. KDs should maintain information about the sharedness of their tickets. Note that keeping a use count for shared tickets only gives you the maximum number of remaining uses. Also note that if a ticket is used to create a shared derivative ticket, the first ticket automatically becomes shared as well.

If public key cryptography is not used, then the checksum in an original ticket T1 is encrypted with the secret key K1 that the KD creating the ticket has for the lock. Additionally, a new secret key K2 (preferably a Combined key as per Bluetooth specification) is created and shared by the KD creating the ticket (KD1) and the KD receiving the ticket (KD2). Instead of the public key of the receiver, KD1 includes in the ticket K2, encrypted with K1. KD2 stores the K2 together with the ticket. The LD can then verify the ticket by decrypting the checksum with the secret key it has stored for the KD1 (K1), and verifying that the checksum is correct. The LD can further verify that KD2 is the KD the ticket was created for, by similarly decrypting K2 with K1, and issuing an authentication challenge as per Bluetooth specification to KD2, using K2 as the link key.

If public key cryptography is not used, then derivative tickets are created as follows: Assume KD2 from previous paragraph wishes to create a derivative ticket T2 for another KD (KD3). It proceeds otherwise as with public key cryptography, but encrypts the checksum using the secret key K2 it has stored for its own ticket. A new secret key K3 is then created and shared by KD2 and KD3 (preferably a Combined key as per Bluetooth specification). KD2 encrypts K3 with K2, and includes it in the new ticket instead of the public key of the receiver. When the LD verifies the ticket, it can obtain K2 from the original (nested) ticket as explained in the previous paragraph. It can then decrypt K3, and

authenticate KD3 as per Bluetooth specification, using K3 as the link key. In this way, tickets can be nested arbitrarily deep.

Below is a description of tickets in Backus-Naur Form notation. It assumes public key cryptography is used. Note that on derivative tickets, the term "granter" refers to the immediate granter (who has a ticket), not the original one (who has a key).

<Ticket> := <Original_ticket> I <Derivative_ticket>

<Original_ticket> := <LID> <TID> <Granter_KID> <Link_key>
<Receiver_public_key> <Access_limits>
<Max_derivation_level> <Derivation_limits>
<Checksum>

<LID> := <LID> of the LD the ticket can open.

<TID> := Unique ticket identifier.

<Granter_KID> := KID of the granter's KD.

<Link_key> := A link key created by applying a well-known secure one way hash function (e.g. MD5 or SH-1) on the <LID> and the link key that is stored on this LD for the granter's KD.

<Receiver_public_key> := The public key of the KD of the receiver of the ticket.

<Access_limits> := Limitations on when the ticket is valid. For example, a time period, or a counter that says how many times this ticket can be used.

<Max_derivation_level> := How many levels of derivative tickets can be created. For example: A <Max_derivation_level> of zero forbids derivative tickets entirely. A level of two allows the receiver of a derivative ticket, that was made by the receiver of an original ticket, still create derivative tickets, but those tickets cannot be used to make derivative tickets. A special value such as -1 can be used to allow infinite levels of derivative tickets.

<Derivation_limits> := **Limits on what kind of derivative tickets may be created. For example, a ticket may be limited so that it only allows creation of one-day derivative tickets.**

<Checksum> := **A secure checksum (e.g. MD5) of all the other information in the ticket (excluding nested tickets), encrypted with the secret key of the granter's KD.**

<Derivative_ticket> :=<LID> <TID> <Receiver_public_key> <Access_limits> <Max_derivation_level><Derivation_limits> <Checksum> <Ticket>

Figure 3 shows an embodiment of the process of using a ticket, from the KD point of view, as a flowchart. Figures 4 and 5 show embodiments of the process of using a ticket, from the LD point of view, as a flowchart

1) The LD broadcasts its service (2020,1020).

2) The KD sends its KID (1030,2030) and the LD replies with its LID, Confirm flag and a flag that tells if the LD knows the KID (false in this case) (2040, 2050, 1040, 1050).

3) The KD finds that it has a valid ticket for the LD (1200). If the confirm flag was true (1210), the KD signals a confirmation request to the user (1220) and awaits confirmation.

4) The KD tells the lock the KID of the granter of the (if nested, innermost) ticket (3030). It authenticates the LD with the ticket link key according to Bluetooth specification (KD challenges, LD responds) (3030, 4030-4080, 3040, 3050). The LD finds the granter's link key based on the granter KID (4040), and can then create the required link key by applying the hash function (4050).

5) Encryption based on the link key is taken into use (3060, 4090). The LD sends a random number to the KD (4100, 3070).